一 前言

验证码是根据随机字符生成一幅图片，然后在图片中加入干扰象素，用户必须手动填入，防止有人利用机器人自动批量注册、灌水、发垃圾广告等等 。

验证码的作用是验证用户是真人还是机器人。

二 背景介绍

先看下本项目的验证码图片

0170.png

本项目验证码图片像素大小为 24*64，由 4 位阿拉伯数字组成，这里图片大小需要关注下，涉及到后面 CNN 模型里相关参数的调整。
首先准备好大约 2000 张左右 24*64 大小的验证码图片训练集，训练集的文件名就是对应图片的标签，如图 1 所示：

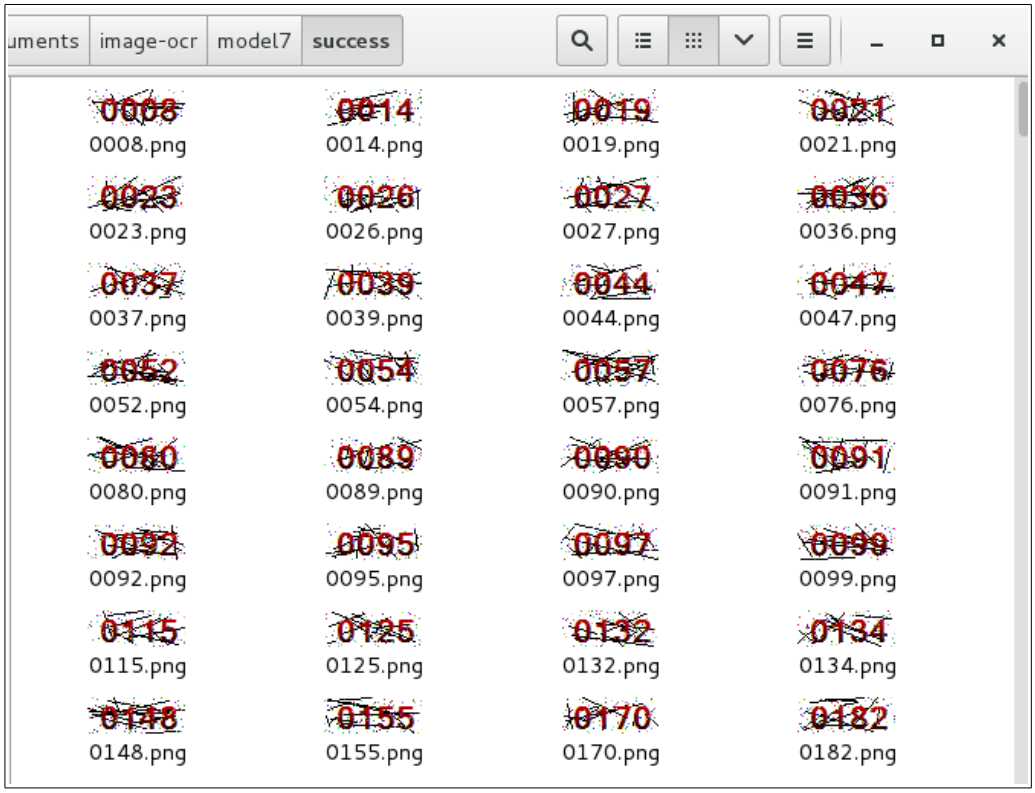

图 1 训练集数据

然后用于模型测试的测试集如图 2 所示，这里我为了便于观察，验证效果，自己手注了 200 张测试集标签，当然可以不用标注，用程序对图片自动随机命名即可：

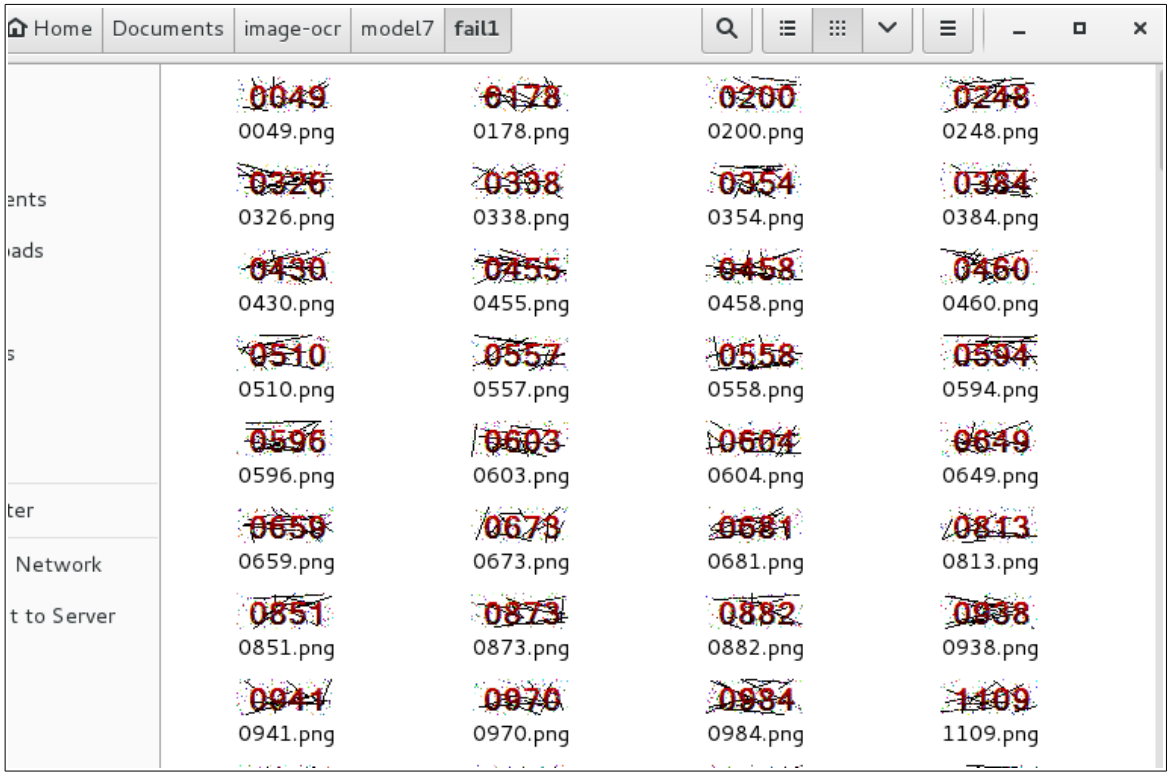

图 2 测试集数据

三 验证码识别

1 数据预处理

首先，数据预处理分为两个部分，第一部分是读取图片，并划分训练集和测试集。训练集大概 2000 张左右的验证码图片，测试集数量大概 200 张（数量随便)。随后，虽然标签是文件名，我们认识，但是机器是不认识的，因此我们要使用 text2vec，将标签进行向量化。

读取数据：

```
    def get_imgs_train(self):
        train_imgs=os.listdir(self.train_data_path)
        random.shuffle(train_imgs)
        #imgs_num_test=len(imgs_test)
        #根据文件名获取训练集标签
        train_labels=list(map(lambda x: x.split('.')[0],train_imgs))
        return train_imgs, train_labels
    def get_imgs_test(self):
        test_imgs=os.listdir(self.test_data_path)
        random.shuffle(test_imgs)
        #imgs_num_test=len(imgs_test)
        #根据文件名获取测试集标签
        test_labels=list(map(lambda x: x.split('.')[0],test_imgs))
        return test_imgs, test_labels
```

2 标签向量化：

既然需要将标签向量化，那么，我们也需要将向量化的标签还原回来。

```
def text2vec(self, text):
    """
    文本转向量
    Parameters:
        text:文本
    Returns:
        vector:向量
    """
    '''
    if len(text) > 4:
        raise ValueError('验证码最长4个字符')
    '''

    vector = np.zeros(4 * self.char_set_len)
    def char2pos(c):
        if c =='_':
            k = 62
            return k
        k = ord(c) - 48
        if k > 9:
            k = ord(c) - 55
            if k > 35:
                k = ord(c) - 61
                if k > 61:
                    raise ValueError('No Map')
        return k
    for i, c in enumerate(text):
        idx = i * self.char_set_len + char2pos(c)
        vector[idx] = 1
    return vector
```

```python
def vec2text(self, vec):
    """
    向量转文本
    Parameters:
        vec:向量
    Returns:
        文本
    """
    char_pos = vec.nonzero()[0]
    text = []
    for i, c in enumerate(char_pos):
        char_at_pos = i #c/63
        char_idx = c % self.char_set_len
        if char_idx < 10:
            char_code = char_idx + ord('0')
        elif char_idx < 36:
            char_code = char_idx - 10 + ord('A')
        elif char_idx < 62:
            char_code = char_idx - 36 + ord('a')
        elif char_idx == 62:
            char_code = ord('_')
        else:
            raise ValueError('error')
        text.append(chr(char_code))
    return "".join(text)
```

3 根据 batch_size 获取数据

我们在训练模型的时候，需要根据不同的 batch_size"喂"数据。这就需要我们写个函数，从整体数据集中获取指定 batch_size 大小的数据。

```
def get_next_batch(self, train_flag=True, batch_size=100):
    """
    获得batch_size大小的数据集
    Parameters:
        batch_size:batch_size大小
        train_flag:是否从训练集获取数据
    Returns:
        batch_x:大小为batch_size的数据x
        batch_y:大小为batch_size的数据y
    """
    # 从训练集获取数据
    if train_flag == True:
        if (batch_size + self.train_ptr) < self.train_size:
            trains = self.train_imgs[self.train_ptr:(self.train_ptr + batch_size)]
            labels = self.train_labels[self.train_ptr:(self.train_ptr + batch_size)]
            self.train_ptr += batch_size
        else:
            new_ptr = (self.train_ptr + batch_size) % self.train_size
            trains = self.train_imgs[self.train_ptr:] + self.train_imgs[:new_ptr]
            labels = self.train_labels[self.train_ptr:] + self.train_labels[:new_ptr]
            self.train_ptr = new_ptr

        batch_x = np.zeros([batch_size, self.heigth*self.width])
        batch_y = np.zeros([batch_size, self.max_captcha*self.char_set_len])

        for index, train in enumerate(trains):
            img = np.mean(cv2.imread(self.train_data_path + train), -1)
             # 将多维降维1维
            batch_x[index,:] = img.flatten() / 255
        for index, label in enumerate(labels):
            batch_y[index,:] = self.text2vec(label)
```

```
    # 从测试集获取数据
    else:
        if (batch_size + self.test_ptr) < self.test_size:
            tests = self.test_imgs[self.test_ptr:(self.test_ptr + batch_size)]
            labels = self.test_labels[self.test_ptr:(self.test_ptr + batch_size)]
            self.test_ptr += batch_size
        else:
            new_ptr = (self.test_ptr + batch_size) % self.test_size
            tests = self.test_imgs[self.test_ptr:] + self.test_imgs[:new_ptr]
            labels = self.test_labels[self.test_ptr:] + self.test_labels[:new_ptr]
            self.test_ptr = new_ptr

        batch_x = np.zeros([batch_size, self.heigth*self.width])
        batch_y = np.zeros([batch_size, self.max_captcha*self.char_set_len])

        for index, test in enumerate(tests):
            img = np.mean(cv2.imread(self.test_data_path + test), -1)
             # 将多维降维1维
            batch_x[index,:] = img.flatten() / 255
        for index, label in enumerate(labels):
            batch_y[index,:] = self.text2vec(label)

    return batch_x, batch_y
```

4 CNN 模型

本项目网络模型为：3 卷积层+1 全链接层。对于 CNN(卷积神经网络)，其模型示意图如图 3 所示：

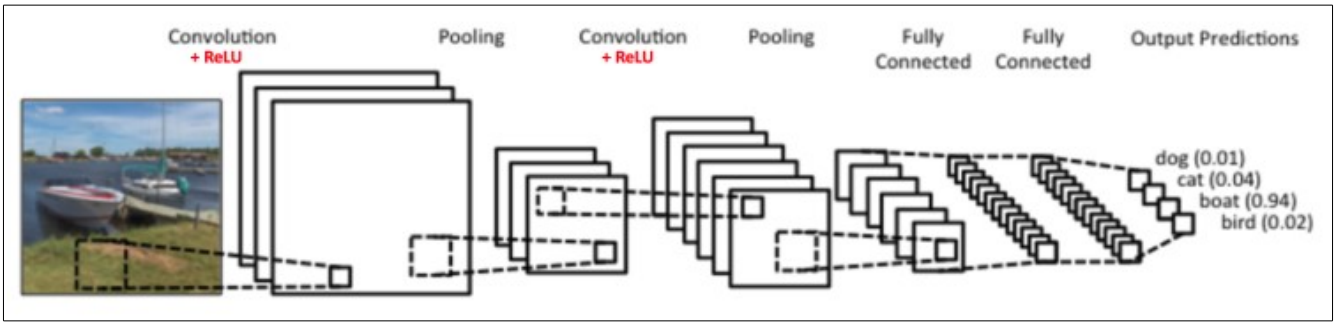

图 3 CNN 模型示意图

```python
def crack_captcha_cnn(self, w_alpha=0.01, b_alpha=0.1):
    """
    定义CNN
    Parameters:
        w_alpha:权重系数
        b_alpha:偏置系数
    Returns:
        out:CNN输出
    """
    # 卷积的input: 一个Tensor。数据维度是四维[batch, in_height, in_width, in_channels]
    # 具体含义是[batch大小, 图像高度, 图像宽度, 图像通道数]
    # 因为是灰度图，所以是单通道的[?, 64, 24, 1]
    x = tf.reshape(self.X, shape=[-1, self.heigth, self.width, 1])
    # 卷积的filter:一个Tensor。数据维度是四维[filter_height, filter_width, in_channels, out_channels]
    # 具体含义是[卷积核的高度, 卷积核的宽度, 图像通道数, 卷积核个数]
    w_c1 = tf.Variable(w_alpha*tf.random_normal([3, 3, 1, 32]))
    # 偏置项bias
    b_c1 = tf.Variable(b_alpha*tf.random_normal([32]))
    # conv2d卷积层输入:
    #   strides: 一个长度是4的一维整数类型数组，每一维度对应的是 input 中每一维的对应移动步数
    #   padding：一个字符串，取值为 SAME 或者 VALID 前者使得卷积后图像尺寸不变，后者尺寸变化
    # conv2d卷积层输出:
    #   一个四维的Tensor, 数据维度为 [batch, out_width, out_height, in_channels * out_channels]
    #   [?, 64, 24, 32]
    #   输出计算公式H0 = (H - F + 2 * P) / S + 1
    #       对于本卷积层而言,因为padding为SAME,所以P为1。
    #   其中H为图像高度,F为卷积核高度,P为边填充,S为步长
    # 学习参数:
    #   32*(3*3+1)=320
    # 连接个数:
    #   64*24*64*24=2359296个连接

    # bias_add:将偏差项bias加到value上。这个操作可以看做是tf.add的一个特例，其中bias是必须的一维。
    # 该API支持广播形式，因此value可以是任何维度。但是，该API又不像tf.add，可以让bias的维度和value的最后一维不同，
    conv1 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(x, w_c1, strides=[1, 1, 1, 1], padding='SAME'), b_c1))
    # max_pool池化层输入：
    #   ksize:池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]
    #       因为我们不想在batch和channels上做池化，所以这两个维度设为了1
    #   strides:和卷积类似，窗口在每一个维度上滑动的步长，一般也是[1, stride,stride, 1]
```

```
    #    padding:和卷积类似，可以取'VALID' 或者'SAME'
    # max_pool池化层输出：
    #    返回一个Tensor，类型不变，shape仍然是[batch, out_width, out_height, in_channels]这种形式
    #    [?, 32, 12, 32]
    # 学习参数:
    #    2*32
    # 连接个数:
    #    12*32*32*(2*2+1)=61440
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    # dropout层
    # conv1 = tf.nn.dropout(conv1, self.keep_prob)
    w_c2 = tf.Variable(w_alpha*tf.random_normal([3, 3, 32, 64]))
    b_c2 = tf.Variable(b_alpha*tf.random_normal([64]))
    # [?, 32, 12, 64]
    conv2 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(conv1, w_c2, strides=[1, 1, 1, 1], padding='SAME'), b_c2))
    # [?, 16, 6, 64]
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    #conv2 = tf.nn.dropout(conv2, self.keep_prob)
    w_c3 = tf.Variable(w_alpha*tf.random_normal([3, 3, 64, 64]))
    b_c3 = tf.Variable(b_alpha*tf.random_normal([64]))
    # [?, 16, 6, 64]
    conv3 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(conv2, w_c3, strides=[1, 1, 1, 1], padding='SAME'), b_c3))
    # [?, 8, 3, 64]
    conv3 = tf.nn.max_pool(conv3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    #conv3 = tf.nn.dropout(conv3, self.keep_prob)
    # [3328, 1024]
    w_d = tf.Variable(w_alpha*tf.random_normal([3*8*64, 1024]))
    b_d = tf.Variable(b_alpha*tf.random_normal([1024]))
    # [?, 3328]
    dense = tf.reshape(conv3, [-1, w_d.get_shape().as_list()[0]])
    # [?, 1024]
    dense = tf.nn.relu(tf.add(tf.matmul(dense, w_d), b_d))
    dense = tf.nn.dropout(dense, self.keep_prob)
    # [1024, 63*4=252]
    w_out = tf.Variable(w_alpha*tf.random_normal([1024, self.max_captcha*self.char_set_len]))

    b_out = tf.Variable(b_alpha*tf.random_normal([self.max_captcha*self.char_set_len]))
    # [?, 252]
    out = tf.add(tf.matmul(dense, w_out), b_out)
    # out = tf.nn.softmax(out)
    return out
```

以上代码展示了如何搭建卷积神经网络，今后若要根据具体项目网站的验证码图片进行识别，只需将验证码图片的输入向量大小进行修改，对于本项目卷积层第一层输入为【？，64，24，32】，？表示输入图片数量不固定，经过第一层卷积输出【？，32，12，32】，第二层卷积输入为【？，32，12，64】，输出为【？，16，6，64】，第三层卷积输入为【？，16，6，64】，输出为【？，8，3，64】，这里对不同的验证码项目需要修改下面这行代码：

w_d = tf.Variable(w_alpha*tf.random_normal([**3*8*64**, 1024]))，这里红色部分需要根据上面的三层卷积后图片输入向量的变化规律作相应修改（不同网站的验证码图片大小不同，作相应修改即可）。

5 训练函数

准备工作都做好了，我们就可以开始训练了。

```
def train_crack_captcha_cnn(self):
    """
    训练函数
    """
    output = self.crack_captcha_cnn()

    # 创建损失函数
    # loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=self.Y))
    diff = tf.nn.sigmoid_cross_entropy_with_logits(logits=output, labels=self.Y)
    loss = tf.reduce_mean(diff)
    tf.summary.scalar('loss', loss)

    # 使用AdamOptimizer优化器训练模型，最小化交叉熵损失
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

    # 计算准确率
    y = tf.reshape(output, [-1, self.max_captcha, self.char_set_len])
    y_ = tf.reshape(self.Y, [-1, self.max_captcha, self.char_set_len])
    correct_pred = tf.equal(tf.argmax(y, 2), tf.argmax(y_, 2))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
    tf.summary.scalar('accuracy', accuracy)

    merged = tf.summary.merge_all()
    saver = tf.train.Saver()
    with tf.Session() as sess:
        # 写到指定的磁盘路径中
        train_writer = tf.summary.FileWriter(self.log_dir + '/train', sess.graph)
        test_writer = tf.summary.FileWriter(self.log_dir + '/test')
        sess.run(tf.global_variables_initializer())

        # 遍历self.max_steps次
        for i in range(self.max_steps):
            # 迭代500次，打乱一下数据集
            if i % 499 == 0:
                self.test_imgs, self.test_labels=self.get_imgs_test()
                self.train_imgs, self.train_labels = self.get_imgs_train()
```

```
            # 每10次，使用测试集，测试一下准确率
            if i % 10 == 0:
                batch_x_test, batch_y_test = self.get_next_batch(False, 100)
                summary, acc = sess.run([merged, accuracy], feed_dict={self.X: batch_x_test, self.Y: batch_y_test, self.keep_p
                print('迭代第%d次 accuracy:%f' % (i+1, acc))
                test_writer.add_summary(summary, i)

                # 如果准确率大于95%，则保存模型并退出。
                if acc > 0.95:
                    train_writer.close()
                    test_writer.close()
                    saver.save(sess, "/home/apps/model7/model/"+"crack_capcha.model", global_step=i)
                    break
            # 一直训练
            else:
                batch_x, batch_y = self.get_next_batch(True, 100)
                loss_value, _ = sess.run([loss, optimizer], feed_dict={self.X: batch_x, self.Y: batch_y, self.keep_prob: 1})
                print('迭代第%d次 loss:%f' % (i+1, loss_value))
                curve = sess.run(merged, feed_dict={self.X: batch_x_test, self.Y: batch_y_test, self.keep_prob: 1})
                train_writer.add_summary(curve, i)

        train_writer.close()
        test_writer.close()
        saver.save(sess, "/home/apps/model7/model/"+"crack_capcha.model", global_step=self.max_steps)
```

训练完毕后，会在 model 文件夹下生成训练模型，如图 4 所示：

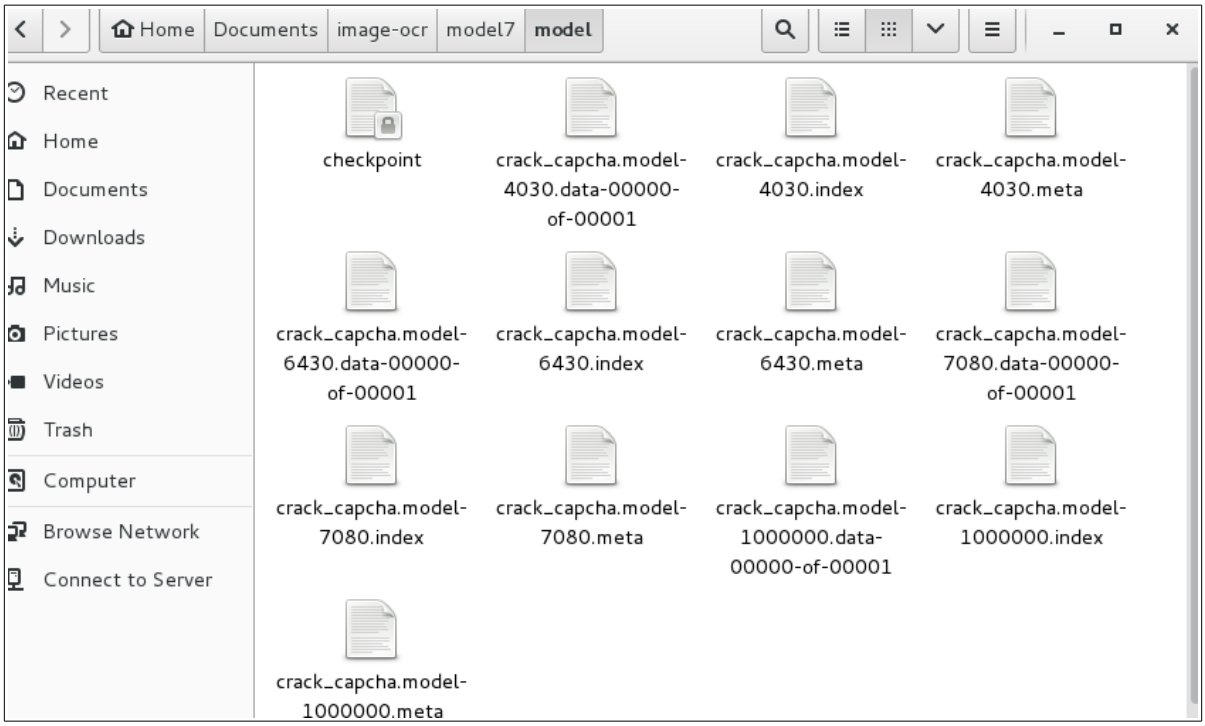

图 4 生成的模型文件

这里可以看到有好多文件，这主要是因为我在训练时有尝试不同的准确率，所以保存的模型文件有好几个相应迭代次数下的文件，最后当准确率达到 95%时，训练结束，此时模型共迭代了 7080 次，大约需要 40-50 分钟，这里训练模型文件我们在测试调用时，所需要的主要就 4 个：checkpoint，以及 crack_capcha.model-1000000 模型的.data,index,meta 三个文件，这 4 个文件即为我们项目所需的训练模型。

6 测试代码

已经有训练好的模型了，怎么加载已经训练好的模型进行预测呢？在和 train.py 相同目录下，创建 test.py 文件，添加如下代码：

```python
import tensorflow as tf
import numpy as np
import train

def crack_captcha(captcha_image, captcha_label):
    """
    使用模型做预测
    Parameters:
        captcha_image:数据
        captcha_label:标签
    """

    output = dz.crack_captcha_cnn()
    saver = tf.train.Saver()
    with tf.Session() as sess:

        saver.restore(sess, tf.train.latest_checkpoint("/home/apps/model7/model"))
        for i in range(len(captcha_label)):
            img = captcha_image[i].flatten()
            label = captcha_label[i]
            predict = tf.argmax(tf.reshape(output, [-1, dz.max_captcha, dz.char_set_len]), 2)
            text_list = sess.run(predict, feed_dict={dz.X: [img], dz.keep_prob: 1})
            text = text_list[0].tolist()
            vector = np.zeros(dz.max_captcha*dz.char_set_len)
            i = 0
            for n in text:
                    vector[i*dz.char_set_len + n] = 1
                    i += 1
            prediction_text = dz.vec2text(vector)
            print("标签: {}  预测: {}".format(dz.vec2text(label), prediction_text))

if __name__ == '__main__':
    dz = train.Ocr()
    batch_x, batch_y = dz.get_next_batch(False, 20)
    crack_captcha(batch_x, batch_y)
```

运行程序，随机从测试集挑选 20 张图片，效果如下：

```
2018-05-07 02:33:34.719518: I tensorflow/core/plat
标签： 4009  预测： 4909
标签： 1192  预测： 1192
标签： 6644  预测： 6644
标签： 0649  预测： 0649
标签： 6240  预测： 6240
标签： 4164  预测： 4164
标签： 2991  预测： 2991
标签： 0049  预测： 0049
标签： 6805  预测： 6805
标签： 9455  预测： 9455
标签： 0813  预测： 0813
标签： 3193  预测： 3193
标签： 1716  预测： 1716
标签： 3653  预测： 3653
标签： 6815  预测： 6815
标签： 5337  预测： 5337
标签： 1562  预测： 1562
标签： 8664  预测： 8664
标签： 3751  预测： 3754
标签： 6828  预测： 6828
```

四 总结

通过修改网络结构，以及超参数，可继续优化本项目。

项目全部代码的 gitlab 地址：http://git.epmap.org/jiwei.chen/OCR